Příloha 10

Kódové provedení mtools

suspicious_intervals – algoritmus pro získání příznaků learning entropy
suspicious_frequencies – algoritmus pro získání příznaků FFT

```
import numpy as np
from STUDNA.ptools import learning_entropy_2, clear_noise_rough,
change_order
from STUDNA.ptools import fourier_transformation, find_fft_peaks,
new_fft_peaks


def suspicious_intervals(learning_effort, time_domain, noise_std: float,
cl_par: float = 1, **kwargs):
    """
    Description tool that uses learning entropy to recognize changes in
signal and
    inform of intervals with suspicious activity.

    From the testing this algorithm is capable of safely detecting changes
in signal that are higher
    than 30% of the noise std. Lower percentage is speculated and requires
bw_mag to be maximum.

    WARNING: When the error is smaller than 30% of the noise, then
possibility of false-negative and
    false-positive rapidly increases.

    Parameters
    --------------------
    learning_effort : array
        Sum of absolute value weights of learning neuron summed up
        trough each epoch.

    time_domain : array
        Domain of observing window

    cl_par : int
        Cluster parameter in time_domain units. Determines what should
algorithm
        recognize as one cluster.

        Example: There are 4 suspicious places in signal with time
difference
        (.3, .5, 1, 2) seconds. If cl_par = 1 (s), then three suspicious
places
        with differences (.3,.5,1) are recognized as one cluster.

    noise_std : float
        Standard deviation of the signal noise. It is important for correct
        sensitivity parameter adjustment.


    Returns
    --------------------
    crit_pos : dict
        Dictionary that contains intervals of unusuality,
```

```
        total number of unusualities and total duration of
        unusuality.
            'intervals' - List of suspicious intervals in tuple (start,
end)
            'sum' - Sum of all suspects
            'duration' - Total duration of unusual intervals
            'mean' - mean of suspected values
            'std' - std of suspected values


    Keyword Arguments
    -------------------
    adjust_mode : bool
        Mode of function which returns more parameters for
        analysis.

        Careful! Instead of 1 return, tuple of 6 parameters is
        returned.

        Returns:
            crit_pos : dict
                Description above.

            time : array
                Due to calculating differentiations time_domain is slightly
                reduced new time is returned.

            crit_pos_bool : array
                Boolean array that corresponds to positions of the
                starts and ends of unusual interval in shorten time domain.

                Useful to compare with pos_diff to control logic of point
                selection.

            suspect_bool : array
                Boolean array of difference conditions in pos_diff. Useful
to compare
                with crit_pos_bool to check algorithm of start/end
decision.

            G : array
                0/1 array that describes what data are selected as unusual.

            pos_diff : array
                Position differences between unusual data (in array G).
Based
                on this array, data are divided into clusters.

                Useful to compare with E_cleaned to see how bw_par
parameter
                changed the selection.

            E_cleaned : array
                Learning entropy that was cleaned by clear_noise_rough()
                PCA cleaner.

                Useful to compare with E to control how cleaner changed
                the selection.

            E : array
                Learning entropy calculated by learning_entropy_2().
```

```
    bw_mag : float
        Bandwidth parameter estimator. Helps to estimate bandwidth that
        determines if learning entropy is suspicious or not.

        bandwidth = bw_mag * std + mean

        The lower the value the higher is sensitivity, but less accurate.
        The higher the value the lower is sensitivity, but more accurate.

        With normal distribution of noise:
        bw_mag = 1 corresponds to aprox. 32% probability that the detected
signal is normal
        bw_mag = 2 ... 5%
        bw_mag = 3 ... 0.3%

    bw_par : float
        Ability to overwrite bandwidth that determines if learning entropy
is suspicious or not.

    buff : int
        Parameter that corresponds to the strength of the
        filter. Buff is the size of one side of the matrix
        that is made from data for PCA process.

        If buff==None then it is selected automatically.

    normalize : bool
        Normalization of the data could enhance and equalize the
        unusualness in signal.
        Default: True

    pca_par : float
        Parameter that changes the filter roughness.
        Default: 0.3

    buff_par : float
        Parameter that changes the filter strength. When buff size is
        selected automatically.b
        Default: 0.05

    """
    # Order is left for potential future updates. Orders higher than 2 are
not tested.
    order = 2

    bw_mag = 1.47 if isinstance(kwargs.get('bw_mag'), type(None)) else
float(kwargs.get('bw_mag'))

    # To detect suspicious intervals we are using learning entropy which
enhances unusualness in
    # learning effort
    E = learning_entropy_2(learning_effort, order=order)

    # Filter will remove negligible unusualness
    E_cleaned = clear_noise_rough(E, buff=kwargs.get('buff'),
normalize=kwargs.get('normalize'),
                                  pca_par=kwargs.get('pca_par'),
buff_par=kwargs.get('buff_par'))

    # Estimate sensitivity by magnitude, mean and noise
```

```
    if isinstance(kwargs.get('bw_par'), type(None)):
        bw_par = bw_mag*np.std(E_cleaned.T**2) + np.mean(E_cleaned.T**2)
    else:
        bw_par = kwargs.get('bw_par')

    # Last separation of negligible and useful unusuality by
    # threshold that corresponds to multiplication of data mean
    # cleaned learning entropy is magnified for better results
    G = E_cleaned.copy() ** 2
    G[G >= bw_par] = 1
    G[G < bw_par] = 0
    # Original
    # G[G >= bw_par * np.mean(G)] = 1
    # G[G < bw_par * np.mean(G)] = 0

    # There is a possibility that we did not made a selection of suspicious
    # intervals. In that case output should be 'nan'
    if sum(G) != 0:
        # Based on G we select time-frames that contain significant
        # unusualities in order to define intervals of unusualities
        pos = time_domain.reshape(-1, 1)[order:][G == 1]

        # We separate unusualities into clusters based on how far away they
are
        # from each other in terms of time steps.
        pos_diff = change_order(pos, 2, False)
        suspects_bool = (pos_diff <= cl_par)
        crit_pos_bool = np.array([False] * len(suspects_bool))

        # Final step is to select beginning and end of each interval
        # [FALSE, TRUE] in array means beginning
        # [TRUE, FALSE] means end of unusuality interval
        prev_state = False
        for k, state in enumerate(suspects_bool):
            if (not prev_state) and state:
                crit_pos_bool[k] = True

            elif prev_state and not state:
                crit_pos_bool[k - 1] = True

            prev_state = state

        # If there is odd number of checkpoints which means one of the
intervals
        # started but has not ended, then end of this interval is probably
last point
        # of the array.
        if sum(crit_pos_bool) % 2 > 0: crit_pos_bool[-1] = True

        starts = pos[crit_pos_bool][::2]
        ends = pos[crit_pos_bool][1::2]
        crit_pos = {'intervals': [(start, end) for start, end in
zip(starts, ends)],
                    'sum': int(sum(G)),
                    'duration': sum(ends - starts[:len(ends)]),
                    'mean': np.mean(E_cleaned[G == 1]),
                    'std': np.std(E_cleaned[G == 1])}

    else:
        pos_diff = np.array([])
        crit_pos_bool = np.array([False])
```

```
        suspects_bool = np.array([False])
        crit_pos = {'intervals': [(float('nan'), float('nan'))],
                    'sum': 0,
                    'duration': float('nan'),
                    'mean': float('nan'),
                    'std': float('nan')}

    if kwargs.get('adjust_mode') == True:
        return crit_pos, time_domain[order:], crit_pos_bool, suspects_bool,
G, pos_diff, E_cleaned, E
    else:
        return crit_pos


def suspicious_frequencies(bench_dataset, compared_dataset, timestep):
    """
    Description tool based on fourier transform. Detects new frequencies in
that
    appeared in the compared_dataset in relation to bench_dataset.

    Parameters
    -------------
    bench_dataset : array-like
        Dataset that contains frequencies that will be ignored.
        Serves as benchmark.

    compared_dataset : array-like
        Dataset that might contain new frequencies.

    timestep : float
        Time difference on time-domain. Serves to calculate the frequency
domain.


    Returns
    ----------
    returns : dict
        Dictionary that contains new appeared power spectral densities
(psd) and
        frequencies (frq). Ordered by psd from highest to lowest.
            'psd' - power spectral density
            'frq' - frequency

    """
    returns = dict()

    # Calculate freq. and psd
    psd_bench, frq_bench = fourier_transformation(bench_dataset, timestep)
    psd, frq = fourier_transformation(compared_dataset, timestep)

    # Find freq.
    _, frqs_bench = find_fft_peaks(psd_bench, frq_bench)
    psds, frqs = find_fft_peaks(psd, frq)

    # Find new frequencies
    returns['psd'], returns['frq'] = new_fft_peaks(frqs_bench, frqs, psds)

    # Sort from highest to lowest
    sorter = {key: value for key, value in zip(returns['psd'],
returns['frq'])}
    sorter = dict(sorted(sorter.items(), reverse=True))
```

```
returns['psd'] = np.array(list(sorter.keys()))
returns['frq'] = np.array(list(sorter.values()))

return returns
```